Analiza abstrakcyjna zmiennych numerycznych oraz struktur danych

Autoreferat rozprawy doktorskiej

Jędrzej Fulara

30 października 2012

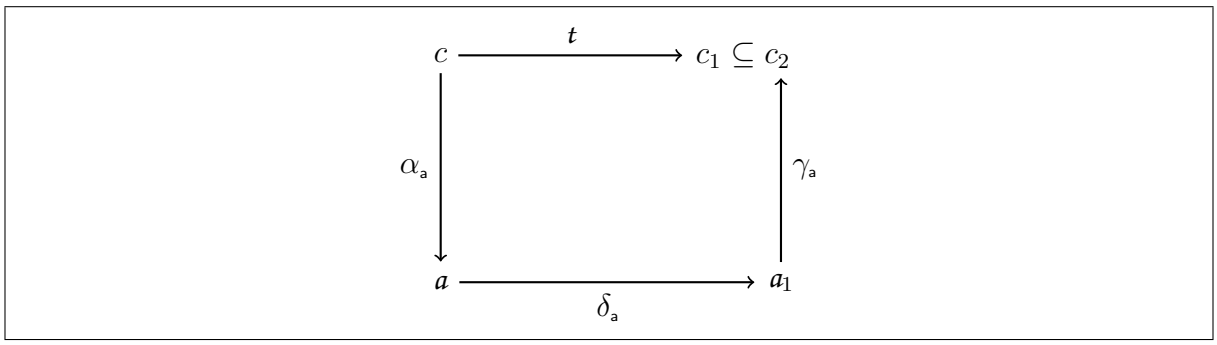
1 Wstęp

Znaczenie systemów komputerowych we współczesnej gospodarce jest trudne do przecenienia. Branża rozrywkowa, administracja publiczna, bankowość czy przemysł ciężki nie mogłyby funkcjonować bez niezawodnego oprogramowania. Firmy informatyczne poświęcają znaczące środki na zapewnienie możliwie wysokiej jakości tworzonych przez nie systemów. Zatrudniają zespoły testerów, których celem jest wykrycie wszelkich błędów w aplikacji, zanim zostanie ona dostarczona klientowi. Testowanie polega na obserwowaniu zachowania aplikacji dla różnych zestawów danych wejściowych. Podejście to ma jedną istotną wadę — z reguły nie da się sprawdzić *wszystkich* scenariuszy, zatem testowanie nie daje gwarancji, że program nie zakończy się błędem w żadnej sytuacji.

Istnieją techniki, za pomocą których można wykazać, że w programie nie występują określone typy błędów. Techniki te, w przeciwieństwie do testowania, działają *statycznie*, to znaczy wnioskują one na temat własności programu jedynie na podstawie jego tekstu i struktury (a nie na podstawie obserwacji wyników jego wykonania). Techniki te mogą zagwarantować, że konkretny typ błędu nie wystąpi w żadnym wykonaniu programu. Z drugiej strony, analiza statyczna może zgłaszać *fałszywe alarmy* — może zasygnalizować potencjalne błędy w całkowicie poprawnym programie.

Analiza statyczna jest najczęściej stosowana do weryfikacji poprawności programów, od których niezawodności zależy ludzkie życie. Została z sukcesem zastosowana między innymi do weryfikacji poprawności kodu sterującego w samolotach firmy Airbus [5, 17], w oprogramowaniu używanym przez Europejską Agencję Kosmiczną [4], czy też w systemach używanych w elektrowniach atomowych [18]. Obecnie podejmowane są próby stworzenia narzędzi pozwalających stosować techniki analizy statycznej do zwykłych programów, na przykład aplikacji pisanych na telefony komórkowe [2].

W ogólnym przypadku nie da się automatycznie stwierdzić, czy dany program nie zawiera błędów. Zatem od analizy statycznej wymagamy jedynie, aby była *poprawna*, czyli by żadne wykonanie pozytywnie przez nią zweryfikowanego programu nie kończyło



Rysunek 1: Zależność między abstrakcją α_a, konkretyzacją γ_a i funkcjami transferu t i δ_a

się błędem. Dopuszczamy jednak, by odrzucała programy w pełni poprawne. Jednym z głównych wyzwań przy tworzeniu nowych technik analizy statycznej jest zapewnienie, by generowały możliwie niewiele takich fałszywych alarmów.

W swojej rozprawie doktorskiej skupiam się na jednym z podejść do analizy statycznej, zwanym *abstrakcyjną interpretacją*.

2 Abstrakcyjna interpretacja

Abstrakcyjna interpretacja jest techniką statycznej analizy programów zaproponowaną w drugiej połowie lat siedemdziesiątych przez Patricka i Radhię Cousot [6].

Celem abstrakcyjnej interpretacji jest obliczenie *przeszacowania* (nadzbioru) zbioru stanów, w jakich program może się znaleźć przy jakichkolwiek danych wejściowych. Jeśli w takim przeszacowaniu nie znajduje się żaden zabroniony stan, to program *na pewno* jest poprawny.

Stan programu opisuje jakie są wartości wszystkich zmiennych używanych w tym programie. Wykonanie programu to ciąg stanów otrzymywanych w wyniku wykonywania kolejnych instrukcji, zaczynając od jakiegoś zestawu danych wejściowych. To jaki stan otrzymamy wykonując daną instrukcję (a więc jak dana instrukcja zmienia wartości zmiennych) jest definiowane przez *funkcję transferu*. Funkcja transferu jest częścią języka programowania. Aby wnioskować na temat różnych własności programów, często wystarcza rozważać zbiory stanów, które mogą się pojawić w konkretnym miejscu w programie w jakimkolwiek wykonaniu [10].

Istotą abstrakcyjnej interpretacji jest zastąpienie prawdziwego zbioru stanów i prawdziwej funkcji transferu zbiorem *stanów abstrakcyjnych* oraz *abstrakcyjną funkcją transferu*. Dodatkowo wprowadzamy funkcje *abstrakcji* oraz *konkretyzacji*, które definiują, jaki zbiór stanów konkretnych jest reprezentowany przez dany stan abstrakcyjny. Zbiór stanów abstrakcyjnych, abstrakcyjna funkcja transferu oraz funkcje abstrakcji i konkretyzacji wchodzą w skład *dziedziny abstrakcyjnej*. Wymagamy, aby dziedzina abstrakcyjna spełniała następujące warunki:

- zbiór stanów abstrakcyjnych $\mathcal{A}$ tworzy kratę zupełną,

- funkcje abstrakcji α_{a} i konkretyzacji γ_{a} spełniają dla każdego stanu abstrakcyjnego a i zbioru stanów konkretnych c zależność $\alpha_{\mathsf{a}}(c) \sqsubseteq_{\mathsf{a}} a \Leftrightarrow c \subseteq \gamma_{\mathsf{a}}(a)$,
- abstrakcyjna funkcja transferu δ_{a} przeszacowuje konkretną funkcję transferu t, co zostało nieformalnie zilustrowane na rysunku 1.

Wynikiem abstrakcyjnej interpretacji zadanego programu jest wektor stanów abstrakcyjnych osiągalnych z abstrakcji wszystkich dopuszczalnych danych wejściowych w każdym punkcie programu. Warunki nałożone na abstrakcyjną dziedzinę gwarantują, że otrzymane w ten sposób stany abstrakcyjne przeszacowują zbiory stanów konkretnych, które mogą się pojawić w danym punkcie programu w każdym wykonaniu rozpoczynającym się od dowolnych dopuszczalnych danych wejściowych.

3 Analiza zmiennych numerycznych

Jednym z podstawowych problemów rozważanych w analizie statycznej jest określenie jakie mogą być wartości poszczególnych zmiennych numerycznych w programie. Zmienne numeryczne są używane (między innymi) przy dostępie do struktur tablicowych. Tak więc, analiza wartości zmiennych numerycznych jest konieczna do sprawdzenia, czy program nie próbuje odczytać elementu z pozycji poza dopuszczalnym zakresem indeksów dla danej tablicy. Wiele numerycznych dziedzin abstrakcyjnych pozwalających znajdować przybliżone rozwiązania tego problemu zostało zaproponowanych. Różnią się one między sobą zarówno precyzją jak i złożonością obliczeniową poszczególnych operacji.

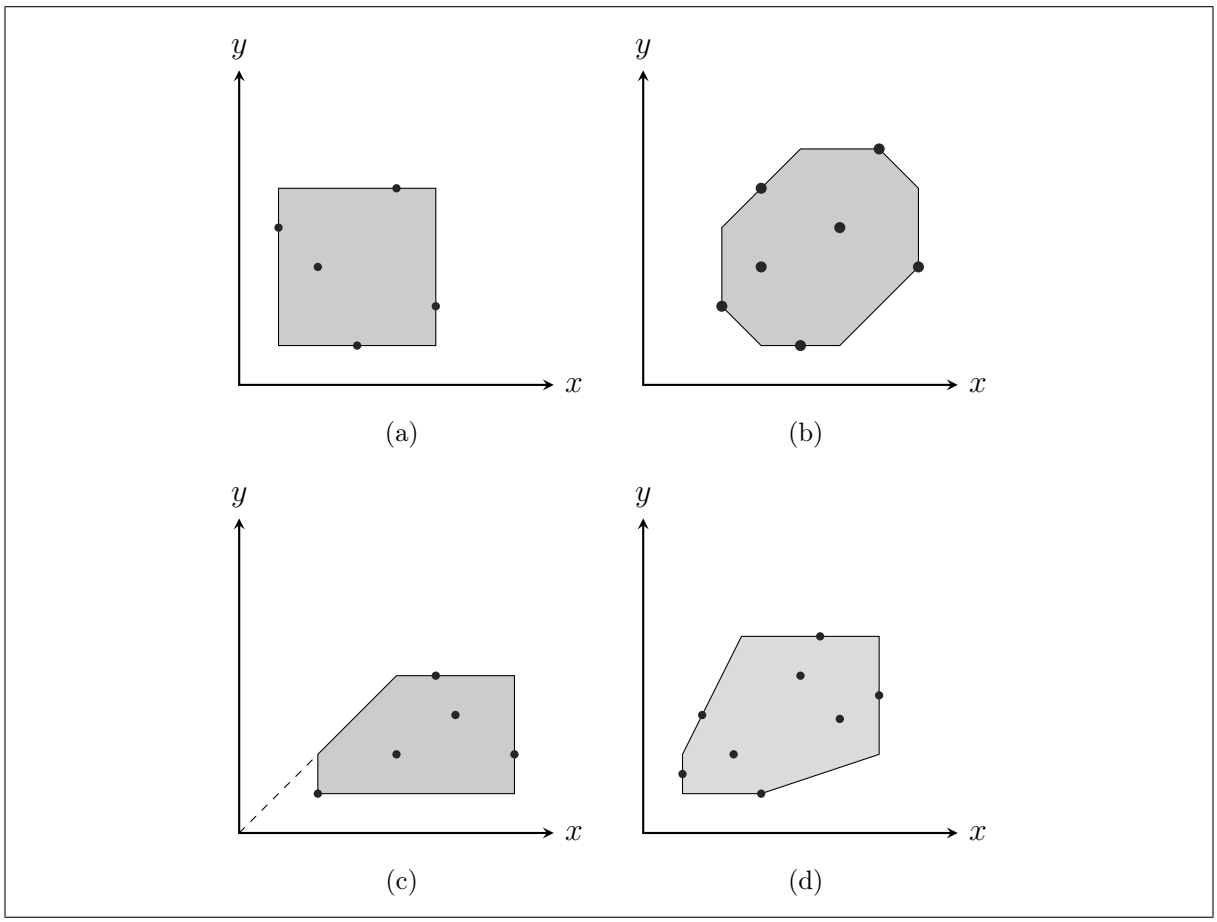
3.1 Dziedzina przedziałowa

Jedną z najprostszych dziedzin numerycznych jest dziedzina przedziałowa [7], w której każda zmienna numeryczna jest reprezentowana przez przedział przeszacowujący zbiór wszystkich możliwych wartościowań tej zmiennej (zatem abstrakcyjny stan w tej dziedzinie to mapa ze zmiennych numerycznych w przedziały ich wartościowań). Główną zaletą tej dziedziny jest jej wydajność — operacje kratowe mają złożoność liniową ze względu na liczbę zmiennych, łatwo jest zdefiniować funkcję transferu. Główną wadą jest brak możliwości wyrażenia jakichkolwiek zależności między zmiennymi, takich jak powiedzenie, że zmienne x i y mogą mieć dowolne wartości, ale zawsze zachodzi nierówność $x \leqslant y$.

Przykładowy element dziedziny przedziałowej został zilustrowany na rysunku 2(a).

3.2 Dziedzina ośmiokątów

Dziedzina ośmiokątów [15] pozwala na wyrażanie zależności postaci $x \pm y \leqslant c$, gdzie x, y oznaczają zmienne w programie, a c jest stałą numeryczną. Dziedzina ta pozwala analizować wartości zmiennych numerycznych z dużo większą precyzją, niż dziedzina przedziałowa. Z drugiej strony, złożoność obliczeniowa podstawowych operacji jest wyższa (sześcienna ze względu na liczbę zmiennych). Abstrakcyjny stan jest reprezentowany jako skierowany graf, w którego wierzchołkach są zmienne, a (etykietowane) krawędzie



Rysunek 2: Elementy dziedziny przedziałów (a), ośmiokątów (b), pięciokątów (c) i ważonych sześciokątów (d) dla dwóch zmiennych.

odpowiadają nierównościom. Sześcienny koszt operacji kratowych wynika z konieczności liczenia najkrótszych ścieżek między wszystkimi parami wierzchołków, co jest realizowane za pomocą algorytmu Floyda-Warshalla.

Dziedzina ośmiokątów zawdzięcza swoją nazwę kształtowi, jaki można ograniczyć układem nierówności postaci $x \pm y \leqslant c$ między parą zmiennych x i y, co zostało zilustrowane na rysunku 2(b).

3.3 Dziedzina pięciokątów

Najprostszą dziedziną relacyjną jest dziedzina pięciokątów [14]. Pozwala ona wyrażać zależności przedziałowe oraz nierówności postaci $x < y$. Dziedzina ta została zaprojektowana przez firmę Microsoft specjalnie do weryfikacji dostępów do tablic. Dzięki ograniczeniu siły wyrazu (w stosunku do dziedziny ośmiokątów), udało się uzyskać niższą (kwadratową) złożoność obliczeniową wszystkich operacji dziedzinowych. Przykładowy element dziedziny pięciokątów (dla dwóch zmiennych) został zaprezentowany na rysunku 2(c).

3.4 Inne dziedziny numeryczne

Poza wspomnianymi trzema dziedzinami, istnieje również wiele innych, o bardzo różnej sile wyrazu. Wśród najprostszych należy wymienić dziedzinę znakową [16] (która dla danej zmiennej potrafi określić jedynie jej znak). Na drugim biegunie znajduje się dziedzina wypukłych wielościanów [9], za pomocą której można wykrywać dowolne nierówności liniowe między zmiennymi. Istotną wadą tej dziedziny jest jej wykładnicza złożoność.

4 Dziedziny ważonych sześciokątów

W rozprawie przedstawiamy dwie nowe abstrakcyjne dziedziny numeryczne. W rozdziale trzecim definiujemy dziedzinę ważonych sześciokątów [12]. Pozwala ona wykrywać ograniczenia przedziałowe oraz zależności między zmiennymi postaci $x \leqslant c \cdot y$, gdzie x, y oznaczają zmienne numeryczne, a c jest nieujemną stałą. Warto zauważyć, że nierówności tej postaci nie mogą być wyrażone w żadnej z omówionych podstawowych dziedzin (mogą być oczywiście wyrażone w technikach bardziej wymagających obliczeniowo, takich jak dziedzina wypukłych wielościanów). Oznacza to, że proponowana przez nas dziedzina pozwala na precyzyjne modelowanie pewnych typów instrukcji (na przykład $x \leftarrow 2 \cdot y$), które w analizie z wykorzystaniem dziedzin ośmiokątów czy pięciokątów mogą być interpretowane jedynie w sposób przybliżony.

Sposób reprezentacji stanów abstrakcyjnych opiera się na obserwacji, że dla każdej pary zmiennych x i y wystarczy trzymać tylko dwie nierówności z zadanego układu omawianej postaci — tę z najmniejszym oraz tę z największym współczynnikiem. Dzięki temu każdy taki układ nierówności możemy zakodować używając dwóch grafów skierowanych, w których wierzchołkami są zmienne programu, a krawędziami są nierówności. Krawędzie są etykietowane odpowiednio najmniejszymi i największymi współczynnikami.

W celu uzyskania optymalnej precyzji operacji kratowych, definiujemy algorytm normalizacji i wykrywania spełnialności stanów abstrakcyjnych. Jest on oparty na uogólnionym algorytmie wyliczania domknięcia przechodniego w grafie [1]. Przeprowadzamy dowód pełności oraz poprawności zaproponowanego algorytmu. Prezentujemy również kompletną funkcję transferu dla prostego języka programowania oraz formalnie pokazujemy poprawność zaprezentowanej dziedziny.

W rozdziale czwartym rozszerzamy dziedzinę ważonych sześciokątów tak, by ujmować również zależności postaci $x < c \cdot y$. To z pozoru niewielkie rozszerzenie, ma potencjalnie duże znaczenie praktyczne — ostre nierówności są bardzo często wykorzystywane w programach, w szczególności w warunkach wejściowych pętli. Automatyczne wykrywanie tego typu zależności może zatem znacząco zwiększyć precyzję analizy.

Należy nadmienić, że istniejące dziedziny abstrakcyjne (włączając w to dziedzinę wypukłych wielościanów) nie pozwalają na jednoczesne wykrywanie ostrych i nieostrych nierówności.

Po rozszerzeniu siły wyrazu naszej dziedziny, struktura stanów abstrakcyjnych staje się nietrywialna: każdy stan abstrakcyjny składa się z trzech etykietowanych grafów skiero-

wanych, przy czym etykiety na krawędziach nie są już czysto numeryczne (co wydatnie komplikuje algorytm normalizacji). Jest to spowodowane koniecznością zakodowania, które nierówności są ostre.

Udało się nam uzyskać sześcienną (ze względu na liczbę zmiennych) złożoność wszystkich operacji. Złożoność pamięciowa reprezentacji stanów abstrakcyjnych jest kwadratowa. Jest to wynik identyczny jak w przypadku dziedziny ośmiokątów.

Rozszerzona wersja dziedziny również została w pełni sformalizowana. Prezentujemy między innymi nietrywialny dowód poprawności algorytmu testowania spełnialności stanów abstrakcyjnych.

Przykładowy element dziedziny ważonych sześciokątów został zilustrowany na rysunku 2(d).

5 Analiza struktur danych

Większość programów używa nie tylko zmiennych numerycznych, ale również struktur danych. Precyzyjne techniki analizy statycznej muszą być wstanie wnioskować na temat zawartości tego typu struktur.

Podstawową strukturą danych jest tablica. Jest to struktura o z góry ustalonej pojemności, do której elementów odwołujemy się podając ich *indeks*. Tablice są indeksowane kolejnymi liczbami naturalnymi, poczynając od zera a kończąc na pewnej z góry ustalonej wartości.

Abstrakcyjna analiza tablic jest dalece bardziej skomplikowana niż analiza zmiennych numerycznych. Po pierwsze, na ogół nie możemy reprezentować każdego elementu tablicy za pomocą oddzielnej wartości abstrakcyjnej — w trakcie analizy możemy nie wiedzieć jaki jest faktyczny rozmiar tablicy. Po drugie, analizując instrukcję modyfikującą tablicę, możemy nie być w stanie określić, który element jest modyfikowany. Aby wyjaśnić ten fakt, rozważmy instrukcję $T[x] \leftarrow 42$, gdzie T oznacza tablicę a x jest zmienną numeryczną. Załóżmy, że do analizy zmiennych numerycznych wykorzystujemy dziedzinę przedziałów i że wiemy, iż przed analizowaną instrukcją $x \in [3, 5]$. Zatem jedyne co możemy o tej instrukcji wywnioskować, to że po jej wykonaniu *któryś* z elementów $T[3]$, $T[4]$ lub $T[5]$ jest równy 42. Jednakże *każdy* z tych elementów może mieć również swoją starą wartość.

Operacje na tablicach są zdefiniowane jedynie dla indeksów z odpowiedniego zakresu (są to zatem operacje częściowe). W swojej pracy rozszerzamy klasyczny formalizm abstrakcyjnej interpretacji tak, by można go było stosować również dla operacji częściowych.

W rozprawie przedstawiamy kilka istniejących technik analizy zawartości tablic [3, 8, 13]. Różnią się one między sobą grupowaniem elementów reprezentowanych przez abstrakcyjne wartości oraz sposobem interpretowania instrukcji modyfikujących tablicę.

Naturalnym rozszerzeniem pojęcia tablicy jest słownik. Jest to struktura danych, w której elementy trzymane są pod kluczami dowolnego typu (w przeciwieństwie do numerycznych indeksów w tablicach). Rozmiar słownika nie jest z góry ustalony i może się

zmieniać podczas wykonania programu. Słowniki pełnią bardzo istotną rolę we współczesnych językach dynamicznie typowanych, takich jak Javascript czy Python.

Wedle naszej wiedzy, w literaturze nie występują dziedziny abstrakcyjne pozwalające analizować zawartość dowolnych słowników. Istniejące techniki analizy zawartości tablic wykorzystują cechy właściwe jedynie tablicom (na przykład numeryczne własności indeksów) i nie da się ich zaadaptować do analizy słowników.

W rozdziale szóstym rozprawy prezentujemy swoją własną technikę, pozwalającą na analizowanie zawartości zarówno tablic jak i dowolnych słowników [11].

6 Abstrakcja tablic i słowników

Zaproponowana przez nas dziedzina abstrakcyjna może zostać użyta do analizy zawartości słowników z kluczami i elementami dowolnych typów. Dziedzina ta jest parametryzowana dwiema innymi dziedzinami abstrakcyjnymi: abstrakcją kluczy i abstrakcją elementów. Każdy słownik jest reprezentowany jako skończony zbiór par (abstrakcyjny klucz, abstrakcyjna wartość). Para taka reprezentuje grupę elementów słownika znajdujących się pod którymkolwiek z kluczy modelowanych przez dany klucz abstrakcyjny. W tym podejściu grupowanie elementów odbywa się w pełni automatycznie i zależy jedynie od własności kluczy. Aby otrzymać jak najlepszą wydajność dziedziny, wprowadzamy wymaganie, by każde dwa abstrakcyjne klucze w danym słowniku były rozłączne. W celu zapewnienia tej własności, definiujemy pojęcia rozłącznego podziału oraz najmniejszego rozłącznego podziału zbioru elementów abstrakcyjnych; dowodzimy, że podziały te zawsze istnieją oraz prezentujemy algorytm ich wyznaczania.

Przedstawiamy wszystkie wymagane elementy dziedziny abstrakcyjnej, w tym strukturę kratową oraz funkcje konkretyzacji i transferu. Pokazujemy, że zaproponowana przez nas konstrukcja spełnia wymagania stawiane przed dziedziną abstrakcyjną.

Definiujemy tak zwane mocne i słabe modyfikacje, co pozwala nam modelować zmiany zawartości słownika możliwie precyzyjnie w każdej sytuacji. Pokazujemy jak wykorzystać silne modyfikacje do precyzyjnego modelowania instrukcji warunkowych.

Prezentujemy przykłady zastosowania naszej techniki do analizy zawartości tablic. Pokazujemy, że siła wyrazu naszego podejścia jest wystarczająca, by na przykład automatycznie zweryfikować poprawność procedury partition z algorytmu quicksort.

Prezentujemy również przykład, w którym nasza dziedzina jest użyta do analizy słowników z kluczami napisowymi. Przykład ten ukazuje potencjalne zastosowanie praktyczne naszej techniki — stosujemy naszą dziedzinę do wykrywania dwóch najczęstszych typów błędów w językach dynamicznie typowanych: odwołania do nieistniejącego atrybutu w obiekcie (`AttributeError`) oraz użycia w operacji obiektu złego typu (`TypeError`).

Omawiana dziedzina została do celów eksperymentalnych zaimplementowana i jej wydajność została sprawdzona na zestawie rzeczywistych, dużych przykładowych programów. Otrzymane rezultaty są porównywalne do tych osiąganych przez istniejące techniki analizy tablic [8], przy czym zakres stosowalności naszego podejścia jest wyraźnie większy.

Literatura

[1] A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

[2] G. Barthe, L. Beringer, P. Crégut, B. Grégoire, M. Hofmann, P. Müller, E. Poll, G. Puebla, I. Stark, and E. Vétillard. Mobius: mobility, ubiquity, security objectives and progress report. In *Proceedings of the 2nd International Conference on Trustworthy Global Computing*, volume 4912 of *LNCS*, pages 10–29, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In T. A. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, *The Essence of Computation*, pages 85–108. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[4] O. Bouissou, E. Conquet, P. Cousot, R. Cousot, J. Feret, K. Ghorbal, E. Goubault, D. Lesens, L. Mauborgne, A. Mine, S. Putot, and X. Rival. Space Software Validation using Abstract Interpretation. In *Int. Space System Engineering Conference, Data Systems in Aerospace DASIA'09*, 2009.

[5] P. Cousot. Proving the absence of run-time errors in safety-critical avionics code. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, EMSOFT '07, pages 7–9, New York, NY, USA, 2007. ACM.

[6] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL'77, pages 238–252, New York, NY, USA, 1977. ACM.

[7] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, pages 237–277. North-Holland, 1977.

[8] P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 105–118, New York, NY, USA, 2011. ACM.

[9] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages*, POPL '78, pages 84–96, New York, NY, USA, 1978. ACM.

[10] R. W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

[11] J. Fulara. Generic abstraction of dictionaries and arrays. *Electron. Notes Theor. Comput. Sci.*, 287C:53–64, 2012.

[12] J. Fulara, K. Durnoga, K. Jakubczyk, and A. Schubert. Relational abstract domain of weighted hexagons. *Electron. Notes Theor. Comput. Sci.*, 267:59–72, October 2010.

[13] D. Gopan, T. Reps, and M. Sagiv. A framework for numeric analysis of array operations. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '05, pages 338–350, New York, NY, USA, 2005. ACM.

[14] F. Logozzo and M. Fähndrich. Pentagons: a weakly relational abstract domain for the efficient validation of array accesses. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 184–188, New York, NY, USA, 2008. ACM.

[15] A. Miné. The octagon abstract domain. *Higher Order Symbolic Computation*, 19(1):31–100, 2006.

[16] M. Sintzoff. Calculating properties of programs by valuations on specific models. *SIGACT News*, (14):203–207, 1972.

[17] J. Souyris, V. Wiels, D. Delmas, and H. Delseny. Formal verification of avionics software products. In *Proceedings of the 2nd World Congress on Formal Methods, FM '09*, volume 5850 of *Lecture Notes in Computer Science*, pages 532–546, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] J. Yoo, S. Cha, and E. Jee. A verification framework for fbd based software in nuclear power plants. In *Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*, pages 385–392, Washington, DC, USA, 2008. IEEE Computer Society.